# Data Structures and Algorithms(UCS540)

Sixth-Semester

Submitted by:

Naman Sood [102104012] 3EE2

BE Third Year (2021-2025) Electrical Engineering

SUBMITTED TO:

MR. YADVENDRA SINGH

Assistant Professor

(Contractual – I)

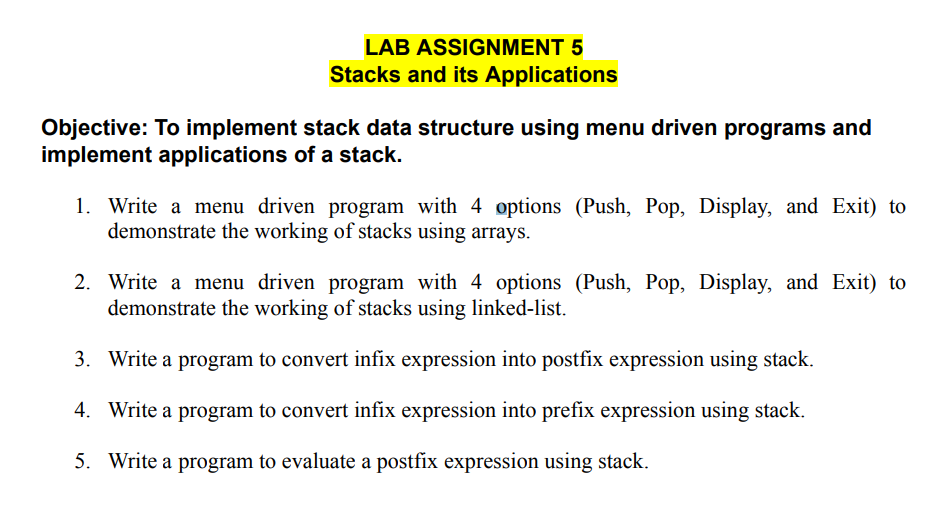


Department of Electrical & Instrumentation Engineering,

Thapar Institute of Engineering & Technology, Patiala

January-May 2024

# List of Experiments



**Q1.**

#include<iostream>

#include<climits>//for INT\_MIN

using namespace std;

template<typename T>

class StackUsingTemplateArrays

{

T\* data;

int nextIndex;

int capacity;

public:

StackUsingTemplateArrays()

{

capacity = 4;

data = new T[capacity];

nextIndex = 0;

}

//return no. of elements in the stack

int size()

{

return nextIndex;

}

bool isEmpty()

{

return nextIndex == 0;//Shortest way to write instead of writing if else statements

}

//insert element

void push(T element)

{

if(nextIndex == capacity)

{

T\* newData = new T[2 \* capacity];

for(int i=0;i<nextIndex;i++)

{

newData[i] = data[i];

}

capacity \*= 2;

delete [] data;

data = newData;

}

data[nextIndex] = element;

nextIndex++;

}

//delete element

T pop()

{

if(isEmpty())

{

cout<<"Stack empty"<<endl;

return 0;

}

else

{

nextIndex--;

T temp = data[nextIndex];

data[nextIndex] = 0;

return temp;

}

}

T top()

{

if(isEmpty())

{

cout<<"Stack is empty"<<endl;

return 0;

}

return data[nextIndex - 1];

}

};

int main()

{

StackUsingTemplateArrays<int> s;

int choice;

while(true) {

cout << "Stack Menu:" << endl;

cout << "1. Push" << endl;

cout << "2. Pop" << endl;

cout << "3. Display top element" << endl;

cout << "4. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch(choice) {

case 1:

int element;

cout << "Enter element to push: ";

cin >> element;

s.push(element);

break;

case 2:

cout << "Popped element: " << s.pop() << endl;

break;

case 3:

cout << "Top element: " << s.top() << endl;

break;

case 4:

cout << "Exiting..." << endl;

exit(0);

default:

cout << "Invalid choice! Please try again." << endl;

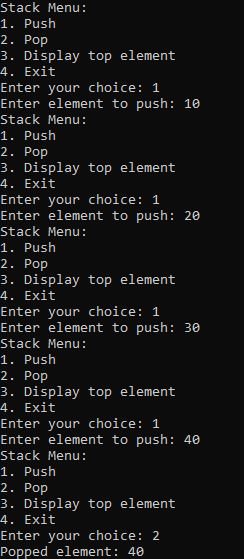
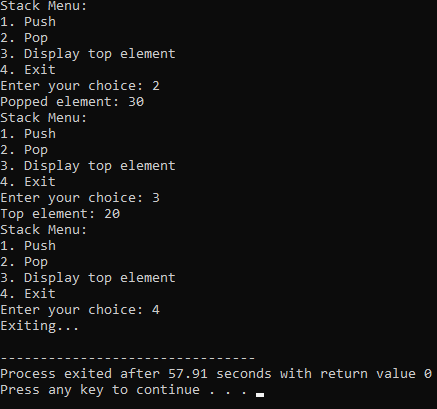
}

}

return 0;

}

**Output:**

**Q2.**

#include<iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data)

{

this->data = data;

this->next = NULL;

}

};

//template<typename T>

class StackUsingLinkedList

{

int stacksize;

Node\* head;

//Node\* top;

public:

StackUsingLinkedList()

{

head = NULL;

//top = NULL;

stacksize = 0;

}

void push(int g)

{

Node\* temp = new Node(g);

temp->next = head;

head = temp;

stacksize++;

}

int pop()

{

if(head == NULL)

{

return -1;

}

int ans = head->data;

Node\* temp = head;

head = head->next;

delete temp;

stacksize--;

return ans;

}

int top()

{

if(head == NULL)

{

return -1;

}

return head->data;

}

int StackSize()

{

return stacksize;

}

bool isEmpty()

{

if(stacksize == 0)

{

return true;

}

else

{

return false;

}

}

};

int main()

{

StackUsingLinkedList stack;

int choice, item;

while (true)

{

cout << "\n---------------------" << endl;

cout << "Stack Implementation using Linked List" << endl;

cout << "---------------------" << endl;

cout << "1. Push" << endl;

cout << "2. Pop" << endl;

cout << "3. Display Top" << endl;

cout << "4. Stack Size" << endl;

cout << "5. Is Empty" << endl;

cout << "6. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter element to push: ";

cin >> item;

stack.push(item);

break;

case 2:

item = stack.pop();

if (item == -1)

cout << "Stack is empty!" << endl;

else

cout << "Popped element: " << item << endl;

break;

case 3:

item = stack.top();

if (item == -1)

cout << "Stack is empty!" << endl;

else

cout << "Top element: " << item << endl;

break;

case 4:

cout << "Stack Size: " << stack.StackSize() << endl;

break;

case 5:

if (stack.isEmpty())

cout << "Stack is empty" << endl;

else

cout << "Stack is not empty" << endl;

break;

case 6:

cout << "Exiting..." << endl;

return 0;

default:

cout << "Invalid choice! Please enter again." << endl;

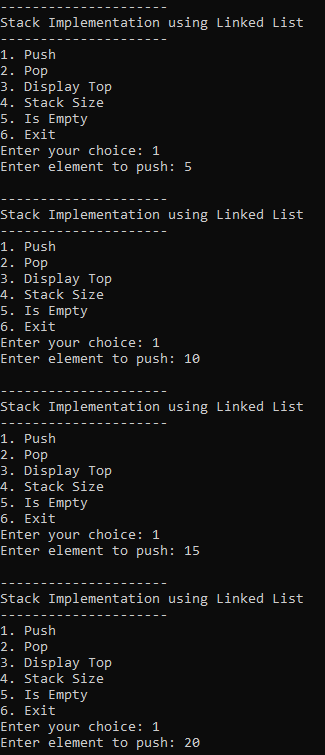
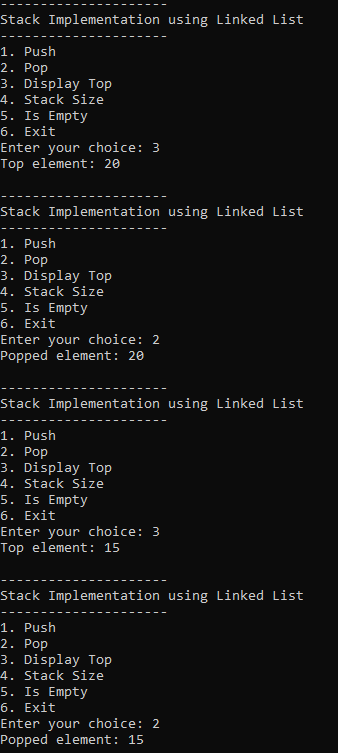
}

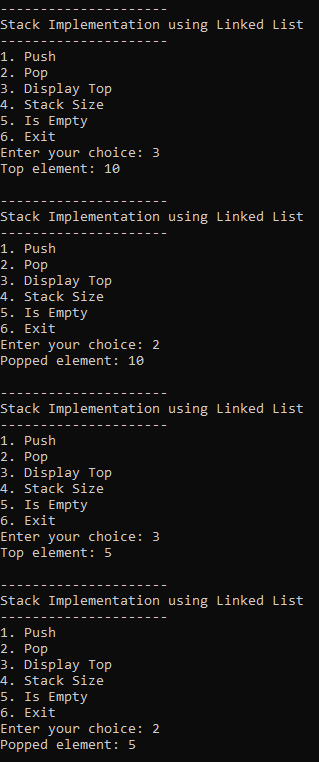
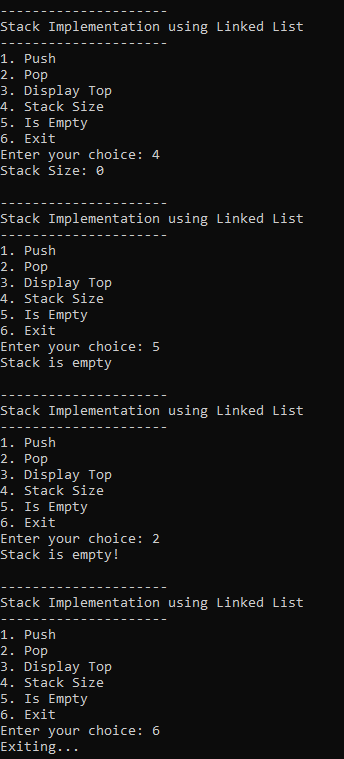
}

return 0;

}

**Output:**

**Q3.**

#include <bits/stdc++.h>

using namespace std;

int precedence(char c) {

if (c == '^')

return 3;

else if (c == '/' || c == '\*')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return -1;

}

char associativity(char c) {

if (c == '^')

return 'R';

return 'L';

}

void infixToPostfix(string s) {

stack<char> st;

string result;

for (int i = 0; i < s.length(); i++) {

char ch = s[i];

if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9'))

result += ch;

else if (ch == '(')

st.push('(');

else if (ch == ')') {

while (st.top() != '(') {

result += st.top();

st.pop();

}

st.pop();

}

else {

while (!st.empty() && precedence(s[i]) < precedence(st.top()) ||

!st.empty() && precedence(s[i]) == precedence(st.top()) &&

associativity(s[i]) == 'L') {

result += st.top();

st.pop();

}

st.push(ch);

}

}

while (!st.empty()) {

result += st.top();

st.pop();

}

cout << "After: " << result << endl;

}

int main() {

string exp = "a+b\*(c^d-e)^(f+g\*h)-i";

cout << "Before: " << exp << endl;

infixToPostfix(exp);

return 0;

}

**Output:**



**Q4.**

#include <iostream>

#include <stack>

using namespace std;

bool isOperator(char c) {

return (!isalpha(c) && !isdigit(c));

}

int getPriority(char C) {

if (C == '-' || C == '+')

return 1;

else if (C == '\*' || C == '/')

return 2;

else if (C == '^')

return 3;

return 0;

}

string reverseString(string str) {

string rev\_str = "";

for (int i = str.size() - 1; i >= 0; i--)

rev\_str += str[i];

return rev\_str;

}

string infixToPostfix(string infix) {

infix = '(' + infix + ')';

int l = infix.size();

stack<char> char\_stack;

string output;

for (int i = 0; i < l; i++) {

if (isalpha(infix[i]) || isdigit(infix[i]))

output += infix[i];

else if (infix[i] == '(')

char\_stack.push('(');

else if (infix[i] == ')') {

while (char\_stack.top() != '(') {

output += char\_stack.top();

char\_stack.pop();

}

char\_stack.pop();

}

else {

if (isOperator(char\_stack.top())) {

if (infix[i] == '^') {

while (getPriority(infix[i]) <= getPriority(char\_stack.top())) {

output += char\_stack.top();

char\_stack.pop();

}

} else {

while (getPriority(infix[i]) < getPriority(char\_stack.top())) {

output += char\_stack.top();

char\_stack.pop();

}

}

char\_stack.push(infix[i]);

}

}

}

while (!char\_stack.empty()) {

output += char\_stack.top();

char\_stack.pop();

}

return output;

}

string infixToPrefix(string infix) {

infix = reverseString(infix);

int l = infix.size();

for (int i = 0; i < l; i++) {

if (infix[i] == '(') {

infix[i] = ')';

} else if (infix[i] == ')') {

infix[i] = '(';

}

}

string prefix = infixToPostfix(infix);

return reverseString(prefix);

}

int main() {

string s = "x+y\*z/w+u";

cout << "Before: " << s << endl;

cout << "After: " << infixToPrefix(s) << endl;

return 0;

}

**Output:**



**Q5.**

#include <iostream>

#include <stack>

using namespace std;

bool isDigit(char c) {

return c >= '0' && c <= '9';

}

int evaluatePostfix(string exp) {

stack<int> stack;

for (int i; i<exp.length(); i++) {

char c = exp[i];

if (isDigit(c))

stack.push(c - '0');

else {

int val1 = stack.top();

stack.pop();

int val2 = stack.top();

stack.pop();

switch (c) {

case '+':

stack.push(val2 + val1);

break;

case '-':

stack.push(val2 - val1);

break;

case '\*':

stack.push(val2 \* val1);

break;

case '/':

stack.push(val2 / val1);

break;

}

}

}

return stack.top();

}

int main() {

string exp = "231\*+9-";

cout << "Postfix evaluation: " << evaluatePostfix(exp) << endl;

return 0;

}

**Output:**

